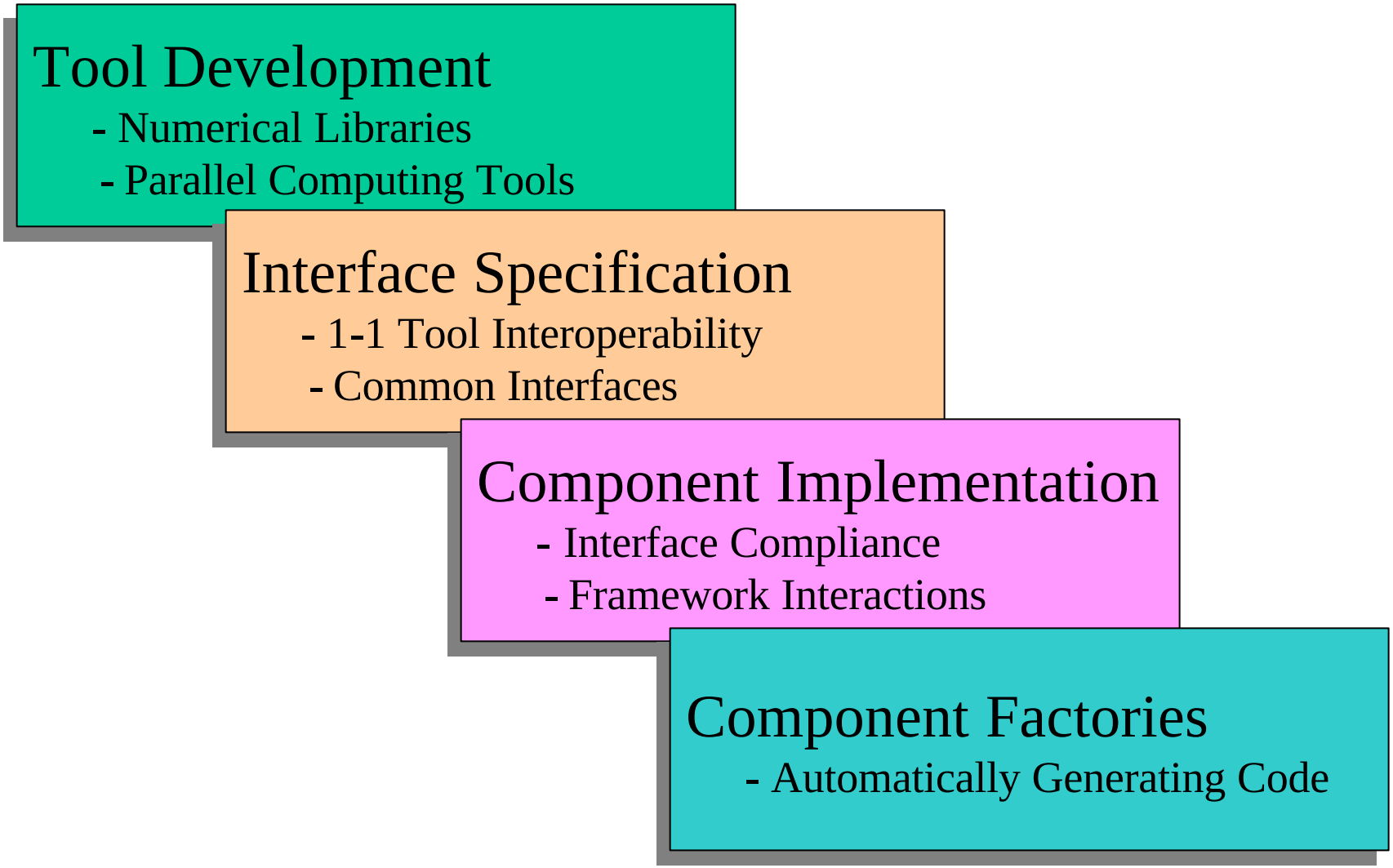
The Development Cycle for Numerical Components: Some Issues and Experiences

Lori Freitag

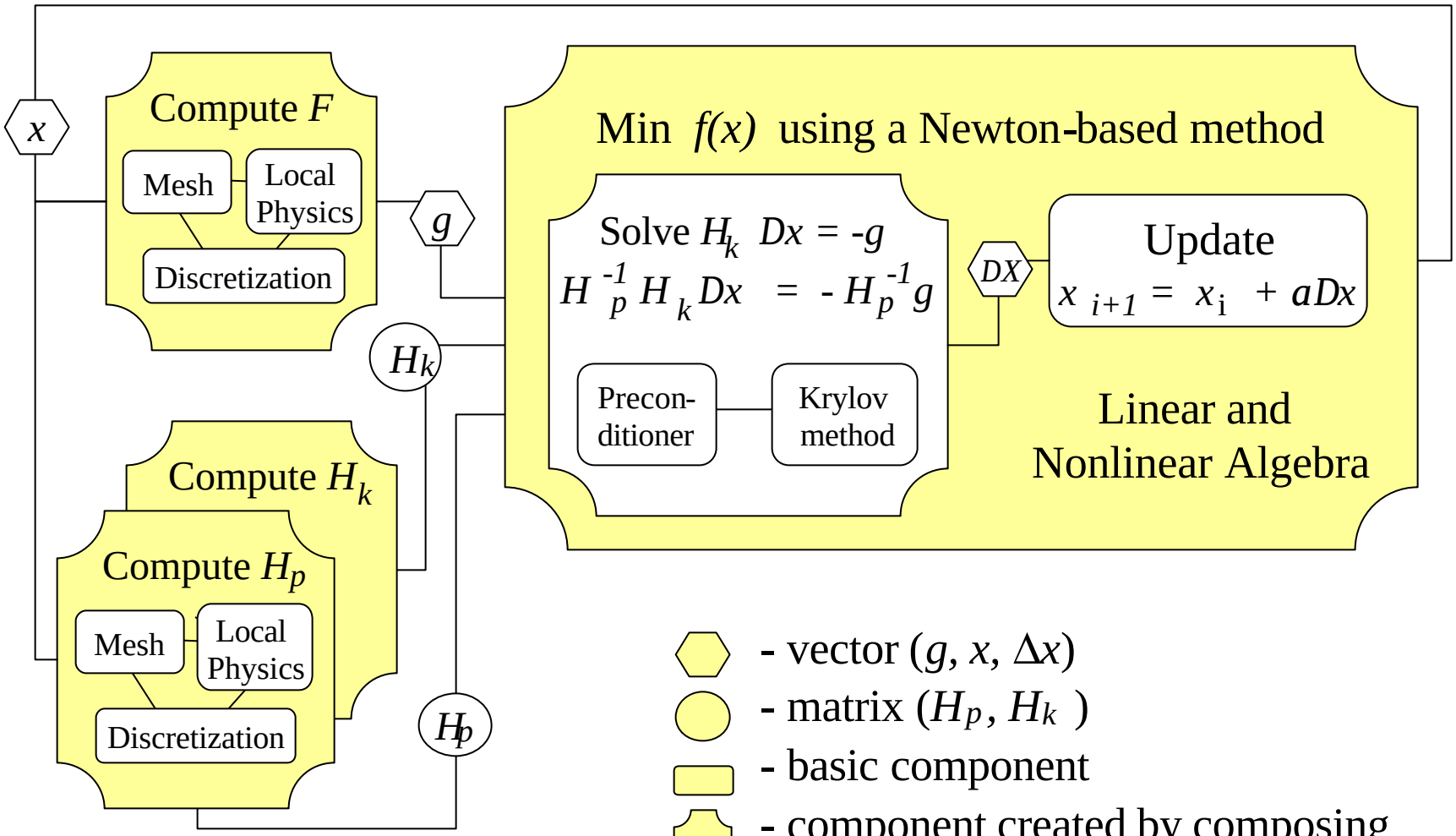
Mathematics and Computer Science Division
Argonne National Laboratory

July 24, 2001

The Component Life Cycle

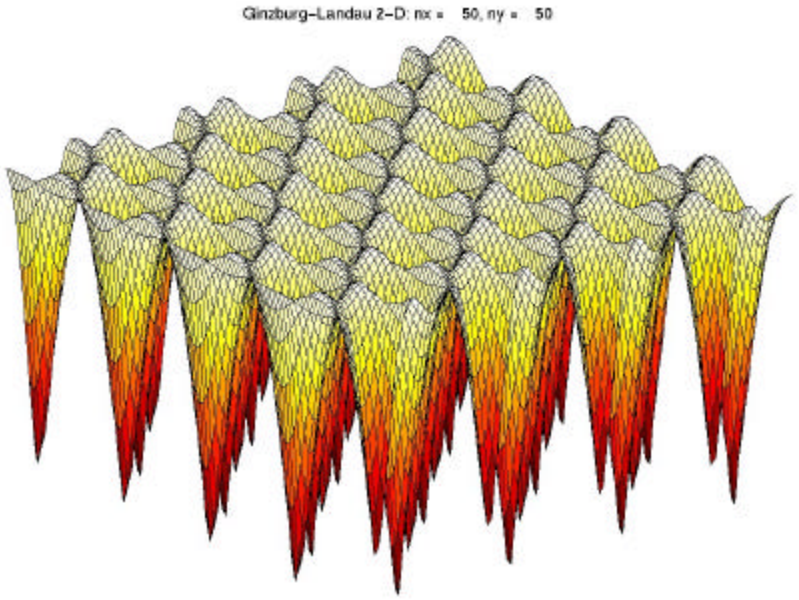


Unconstrained Minimization

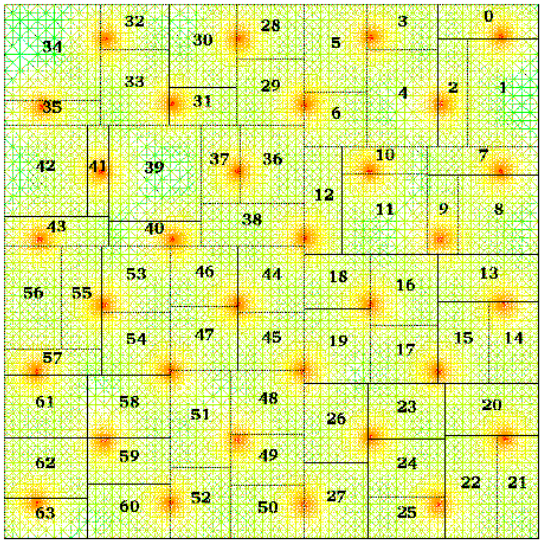


Ginzburg-Landau Superconductivity Model

Unconstrained optimization. Non-convex function. Hessian is singular. Unique minimizer, but there is a saddle point.



Compute F: Structured Mesh, FD
Compute H: ADIFOR (manual)
Solve: TAO and PETSc



Compute F: Unstructured Mesh, FE
Compute H: ADIFOR (manual)
Solve: Custom Newton Solver, BlockSolve

Numerical Tools Used

- **Optimization**
  - TAO
  - Custom Tools
- **Mesh Management**
  - SUMAA3d
  - DAs (as part of PETSc)
- **Linear Solvers**
  - PETSc, BlockSolve95
- **Automatic Differentiation**
  - ADIC, ADIFor
- **Parallel Tools**
  - MPICH, URB Partitioning

Observations:

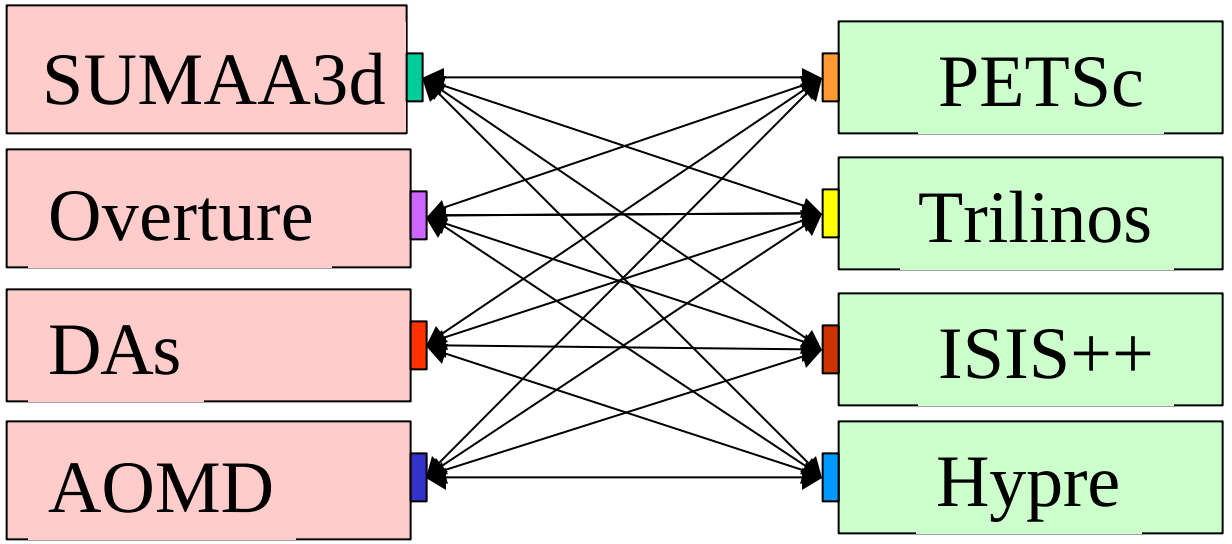
All are worthy

Some Interoperate

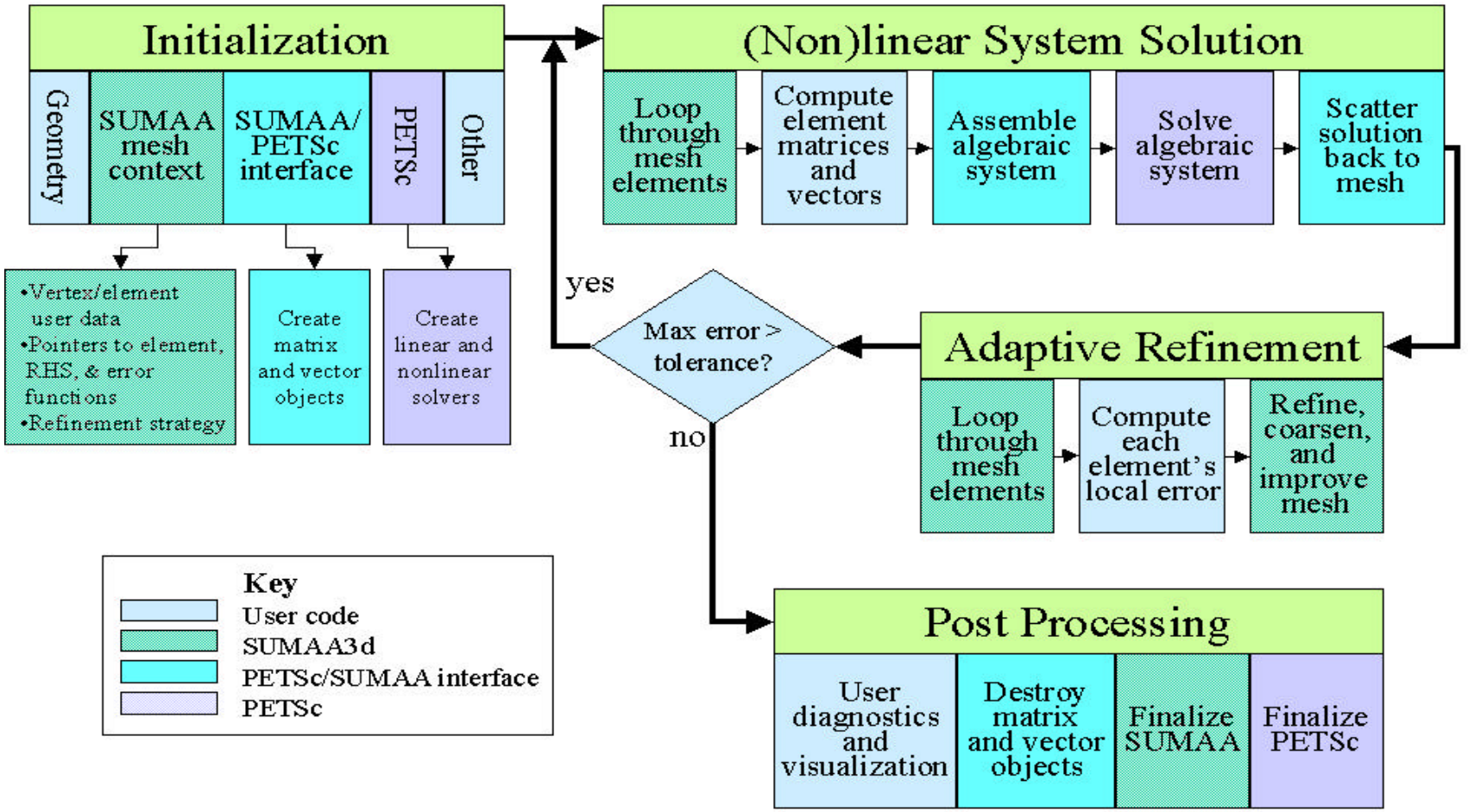
None are Components

Interoperability of Numerical Libraries

- Public interfaces are unique
- *Many-to-Many* couplings require *Many*^{2} interfaces
  - Often a heroic effort to understand the guts of both codes
  - Not a scalable solution



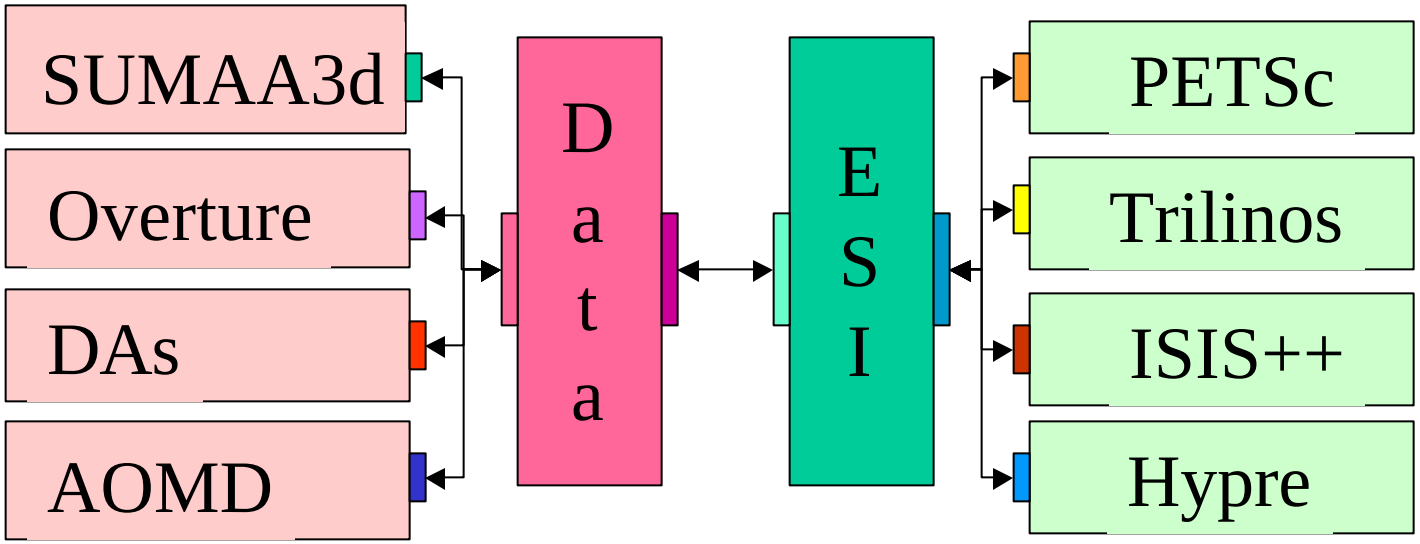
For example…
PETSc-SUMAA3d Interoperability



Common Interface Specification

Reduces the *Many-to-Many* problem to a *Many-to-One* problem

- Allows interchangeability and experimentation
- Difficulties
  - Interface agreement
  - Functionality limitations
  - Maintaining performance



CCA Data Interface Specification

- **Subgroup of CCA forum, meeting quarterly since November 2000**
- **Goals:**
  - To support scientific data access at different levels
    - low-level raw data
    - multi-dimensional arrays
    - meshes and fields
    - parallel data distributions
  - To manage alternative views into a data component via an “interface broker”
  - Intended to be a basic 80-90% solution
- **Various prototype implementations underway**
- **Participants: ANL, UofC, IU, LANL, LLNL, NASA, ORNL, PNNL, SNL, Terascale, UU**

Raw Data Interface

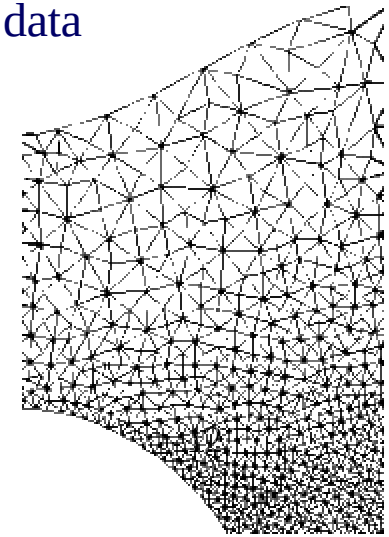
- Set of smart pointers containing name, location, pointer, size, type, strides describing a strip of memory
  - collection of 1D buffers each of which can have a different type
- Modeled on unix IOvecs
- Low level and general
  - can therefore be used as a handle for moving data efficiently
- Considered to be local only
- **Status:** C. Rasmussen proposed, Forum voted and approved

Multi-Dimensional Arrays

- **Proposed Ports**
  - LocalArray
  - LockingLocalArray
  - DataDistQuery
  - DataDistDefine
  - DistributedArray
- **Supports**
  - access to name, dimension, shape, strided subsets
  - various types of distributions (e.g., cyclic, block, mapped)
  - existing packages (e.g., Global Arrays, DAs, A++/P++)
- **Status:**
  - LocalArray and LockingLocalArray have been extensively discussed
  - DataDistQuery, DataDistDefine recently posted for the first time
  - DistributedArray under development
- **Point of Contact:** David Bernholdt (ORNL)

(Unstructured) Meshes and Fields

- **Considerations**
  - Connectivity information is explicitly stored and varies widely
  - Underlying data structures are typically either lists or arrays
  - Many packages exist that will *not* be rewritten
    - Augmented to support a minimal set of required interfaces
- **Status**
  - Nomeclature is determined (node, edge, face, cell, element)
  - Connectivity information available in a table format
  - Basic access to mesh entities via “block iterator” interfaces
    - Provides efficient access to arrays by returning a pointer to the data
    - Provides a natural interface for list-based data structures
    - Provides a mechanism for agglomeration strategies
  - Prototype implementations: Terascale, AOMD, SUMAA3d
  - Testing in real applications to ensure performance
- **Point of Contact:** Lori Freitag (ANL)



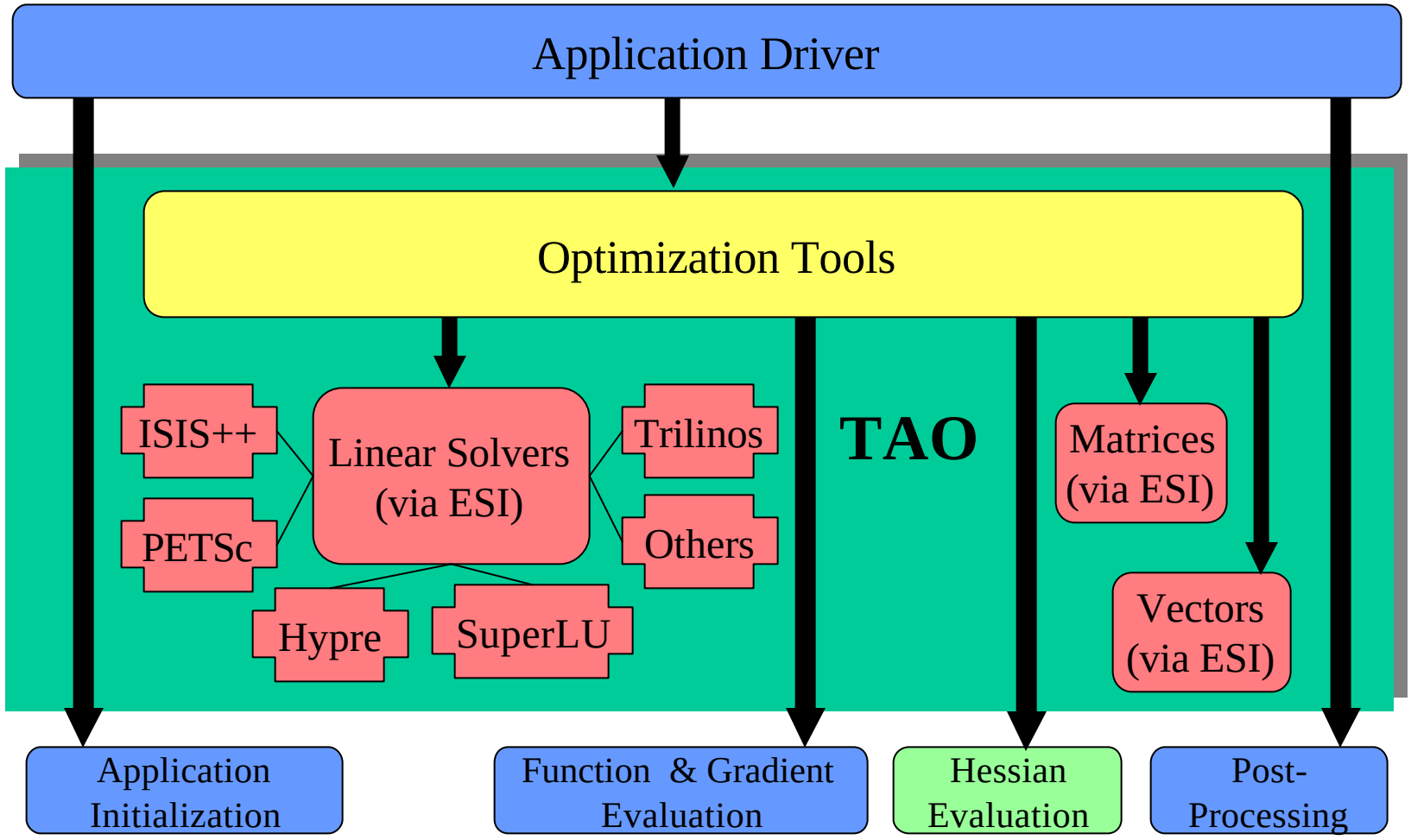
Issues that have arisen...

- **Nomenclature by (an ever-shifting) committee is fun**
- **What level of interface should be defined?**
  - Minimal interfaces
  - Interfaces for convenience and performance?
- **Cannot achieve the 100 percent solution, so...**
  - What functionalities should we define interfaces for?
    - e.g., should the local array group worry about supporting BLAS operations?
  - What about support of existing packages?
    - Are there atomic operations that all support?
    - What additional functionalities from existing packages should be required?
- **Language interoperability issues**
- **Additional functionalities such as locking**

Interface Compliance

- Equation Solver Interface (ESI)
  - Relatively mature interface definition effort
    - Vectors
    - Matrices
    - Linear Solvers
  - Multi-institutional working group
  - Implemented by several solver packages
- Toolkit for Advanced Optimization (TAO)
  - PIs: Benson, McInnes, More'
  - Large-scale, parallel optimization package
    - Unconstrained problems
    - Bound constrained problems
    - Complementarity problems
  - Relies on linear solvers as a kernel computation

TAO


Application Driver
Optimization Tools
ISIS++
PETSc
Linear Solvers (via ESI)
Trilinos
Others
Hypre
SuperLU
TAO
Matrices (via ESI)
Vectors (via ESI)
Application Initialization
Function & Gradient Evaluation
Hessian Evaluation
Post-Processing
User code
TAO code
External linear algebra code
AD-generated code

Original TAO Interface

```
TAO_SOLVER     tao;                /*  optimization solver  */
Vec            x, g;               /*  solution and gradient vectors
PETSc data struct  */
TaoVecPetsc *xx, *gg;              /* wrappers for the PETSc vectors */
ApplicationCtx     usercontext;    /*  user-defined context  */

TaoInitialize();

/* Initialize Application -- Create variable and gradient vectors x and g */
…
vecCreate(MPI_COMM_WORLD, n, &x);

TaoWrapPetscVec(x, &xx);

TaoCreate(MPI_COMM_WORLD,"tao_lmvm",&tao);
TaoSetFunctionGradient(tao,x,g, FctGrad,(void*)&usercontext);

TaoSolve(tao);

/* Finalize application -- Destroy vectors x and g */   ...

TaoDestroy(tao);
TaoFinalize();
```

ESI compliance in TAO

(Leads: Steve Benson, Lois McInnes)

- **TAO originally developed with PETSc vectors**
  - has no “native” TAO data structure for vectors, matrices, linear solvers
  - requires certain functionality that is accessed via wrappers of the underlying supporting packages
- **Migrating to ESI Vectors**
  - Terascale’s implementation as of 11/17/00
  - First implementation uses Vector<double,int>
  - Closely matching functionality (so far) made it fairly straightforward
  - Some additional routines required to support ESI in user functions
- **Difficulties:**
  - Namespace conflict; both PETSc and ESI use the term “Scalar”
  - Not all TAO algorithms can be supported with ESI interfaces
  - Matrices and linear solvers are expected to be more challenging

ESI TAO Interface

```
Include “Vector.h”
TAO_SOLVER     tao;                /*  optimization solver  */
TaoVecESI       *xx;               /*  solution vectors  */
ApplicationCtx     usercontext;    /*  user-defined context  */

TaoInitialize();

MapPartition<int> myMap(MPI_COMM_WORLD, n, 0);    /* ESI vector allocation */
Vector<double,int> *x=new Vector<double,int>(myMap);

xx=new TaoVecESI(x);                                /* wrap the functionality */

TaoCreate(MPI_COMM_WORLD,”tao_lmvm”,&tao);
TaoSetESIFunctionGradient(tao, xx, ESIFctGrad,(void*)&usercontext);

TaoSolve(tao);

/* Finalize application -- Destroy vectors x and g */   ...

TaoDestroy(tao);
TaoFinalize();
```

CCA Compliance in TAO

(Leads: Boyanna Norris, Satish Balay, Lois McInnes)

Paradigm shift; both TAO and the application become components

- Each required to provide a default constructor and to implement the Component interface
  - contains one method: “setServices” to register ports
- All interaction is through ports
  - Application *provides* a “go” port and *uses* “taoSolver” and “MPI” ports
  - TAO *provides* a “taoSolver” port
- There is no “main”

setServices

- Takes CCA Services class as input
  - Provided by the framework
  - Methods: createPortInfo, getPort, releasePort, (un)registerUsesPort, addProvidesPort, removeProvidesPort, getComponentID
- Typical Usage

```
Services *svc;
gov::cca::PortInfo *pi=0
portinfo = svc->createPortInfo("go","gov.cca.GoPort",0);
GoPort *gp = dynamic_cast<gov::cca::GoPort *>(this);
svc->addProvidesPort(gp,portinfo);
```

The Application Go Port

```
TAO_SOLVER     tao;                /*  optimization solver  */
TaoVecESI      *x;                 /*  solution vectors  */
ApplicationCtx usercontext;        /*  user-defined context  */
Port           *p;
TaoSolverComponent *taoSolver

if (scv==0)  return -1;                                  /* check that setServices
was called */

p=svc->getPort((char *) “taoSolver”);                    /* get the TAO Solver
Port */
if (p==0)  return -1;                                    /* check that taoSolver is
out there */
taoSolver = dynamic_cast<TaoSolverComponent *>p;

taoSolver->initialize();

MapPartition<int> myMap(MPI_COMM_WORLD, n, 0);
Vector<double,int> *x=new Vector<double,int>(myMap);
xx=new TaoVecESI(x);

taoSolver->setInitialVector(x);
taoSolver->setGradientVector(g);
taoSolver->create();
taoSolver->setESIFunctionGradient(ESIFctGrad,(void*)&usercontext);
taoSolver->solve();
```

Running the CCA/ESI/TAO example

- Compile each component into a shared library
- Used Sandia's CCAFFIENE as the framework
  - command line interface
- Instantiate each component
  - e.g., create *componentType instanceName* (calls setServices)
  - display component *instanceName* (shows type, name, registered ports)
- Connect the appropriate ports
  - connect *appInstance usesPort solveInstance providesPort*
- Press the go button
  - go *appInstance goPort*

Component Factory Example

(Leads: P. Hovland, L. Grignon, L. McInnes, B. Norris, B. Smith)

- **Automatically generating components from components**
  - Use automatic differentiation to generate gradient and Hessian components from function components
- **What is Automatic Differentiation (AD), you ask?**

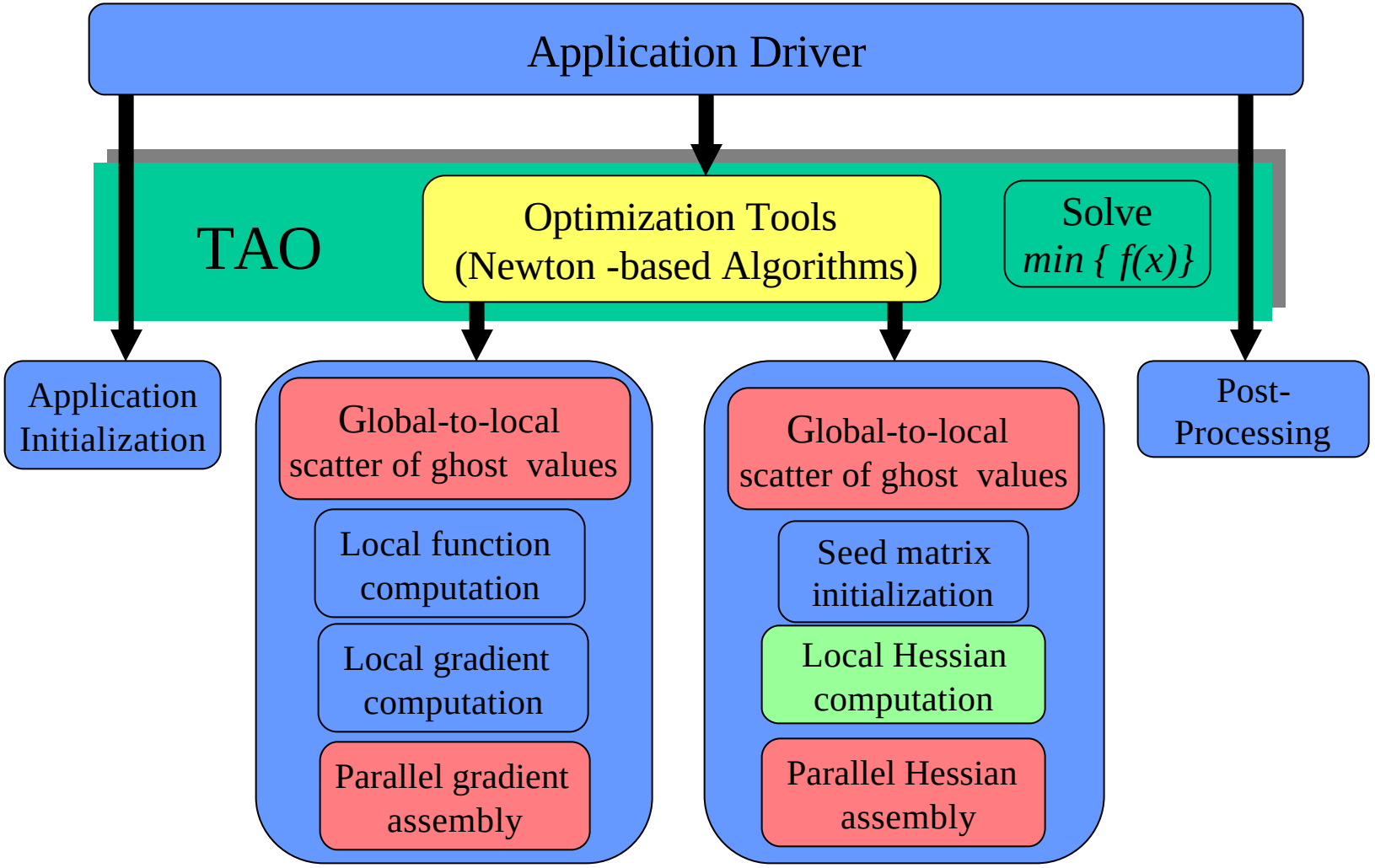
  a technology for automatically augmenting computer programs, including arbitrarily complex simulations, with statements for the computation of derivatives.
  - Every programming language provides a limited number of elementary mathematical functions
  - Every function computed may be viewed as the composition of these so-called intrinsic functions
  - Derivatives for the intrinsic functions are known and can be combined using the chain rule of differential calculus

Why use AD?

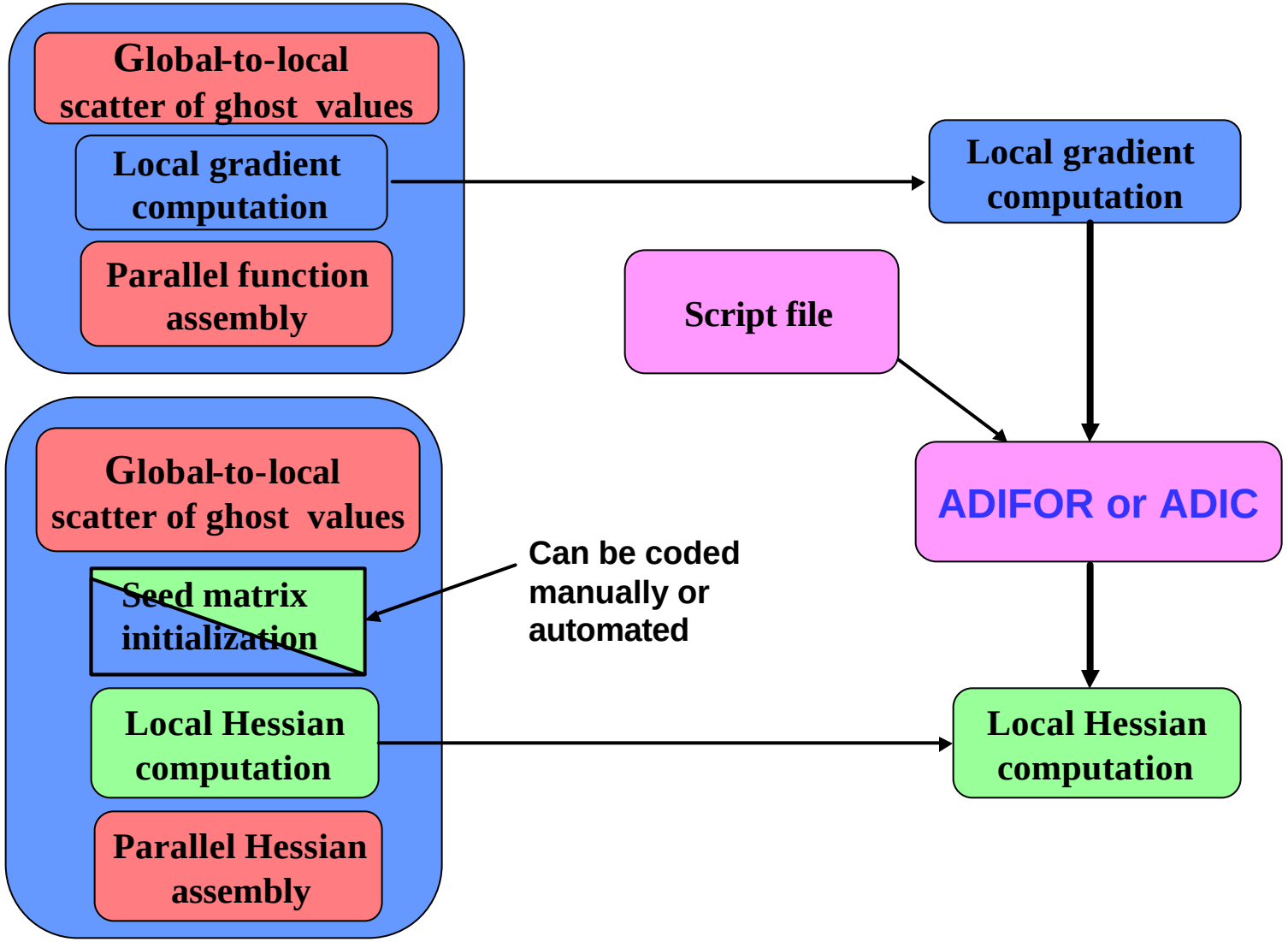
Compared with numerical differentiation via divided differences and hand coding, AD offers

- Reduced effort
- Performance
  - Both FD and AD compute directional derivatives cheaply
    - 1 additional function evaluation for FD
    - ~2 function evaluations for AD (can be less)
- Accuracy
  - FD very sensitive to step size; must balance truncation and roundoff error; guaranteed inaccurate
  - AD analytic - suffers from roundoff error only

Gradient Computations in TAO



Using AD with TAO



Synergies of Component Design and AD

- AD makes numerical software development easier
  - analytic derivatives without handcoding
  - mechanisms for exploiting sparsity
  - cheap directional derivatives
- Component design makes AD easier/better
  - well-defined interfaces enable complete automation of AD
  - components reveal high level semantics, making it possible to exploit mathematics

Tool Availability

- TAO
  - http://www.mcs.anl.gov/tao
- CCA
  - http://www.cca-forum.org
- ESI
  - http://z.ca.sandia.gov/esi/
- PETSc
  - http://www.mcs.anl.gov/petsc
- Automatic Differentiation
  - http://www.mcs.anl.gov/adifor/